Documentation about the WebDAV Interface of the SuSE Linux Openexchange Server

Netline Internet Service GmbH

September 2003

Summary

This document describes the SLOX WebDAV Interface. This documentation gives you a short overview about the basic applications and some development informations. SuSE Linux Openexchange Server is called „SLOX“ inside this document.

Contents

1.Introduction

This document describes the global SLOX WebDAV Interface.
But first of all we should talk about WebDAV. WebDAV is based upon the HTTP Protocol. WebDAV permits to send and receive informations including XML. Well, in fact WebDAV stands for:
"Web-based Distributed Authoring and Versioning".

(If you need further informations about WebDAV, please check the references, that are provided at the end of this document.)

The SLOX WebDAV Interface is a global interface, that includes the most functions of SLOX.
An example: You want to request all contacts from SLOX. Then only those contacts will be returned, what you have the read rights for. The same function is available for all other actions, like update or delete (it is not possible to change/delete an object without the write rights).
If you want to create a task or appointment, then you maybe want to inform all participants (the same handling like in the web interface). This action can also be called with the definition of the SLOX WebDAV Interface.

For all these actions, the client needs to authenticate against SLOX . Authentication must be done by sending username and password information from client to server base64 encoded in the following format:

base64encode(username:password).
(If the authentification fails, SLOX response with a „401 Authorization Required“.)

There are some points in SLOX, that we should explain here for a better understanding the SLOX WebDAV Interface. We should start with the access rights of objects, based on the SLOX access logic:

The objects in SLOX (like appointments, contacts and tasks) have access rights.
With these rights, the creator of the object can define which users or groups can see an object (read rights) or who can edit them (write rights). The SLOX WebDAV Interface returns only those objects, which the logged in user has read rights for. And only objects, which the logged in user has write rights for, can be edited. Every person, who has created or changed an object, is the object owner until the object will be changed by another user, even the creator is not listed in any rights! But there is one exception for this: the „delete right“. If this right is set to one user, the owner of the object will not be changed, until the actual owner defines another owner of the object. And if this right is set, only this owner can delete the object.

If an object is created in a folder, the object inherits the minimal rights from the folder object rights. And these rights can not be deleted. This is very important.

Another point in the SLOX WebDAV Interface is the handling of fields in the XML-Request. Some fields are required (the same as in the webinterface). These fields must be submitted in the requests. All others are optional.

An example:
You want to update a contact and the only field, that you want to change is the telephone number. But you do not want to update any other field. Then you can create an update request with the required fields (for contacts the „lastname“) and add the field „phone“ in the request and send it to SLOX. The result is, that nothing will be changed in the contact, just the name and the phone number. All other fields will be fetched from the existing object (and of course the righs too).
This means, if you send all fields, SLOX will insert/update the given fields. If you only send the required and changed fields, then just those fields will be updated and all other fields and rigths will be taken from the original object.
When you send unkown fields by the SLOX WebDAV Interface, they will be ignored.
But you should remember, that they must be parsed and those fields produce bandwidth and processor-load on your SLOX.

If you want to add participants (appointments and tasks) in your request, please take notes, that only existing users can be added as participants. This means, if you make a request with an unknown participant (no regular user in SLOX), this participant will be automatically removed from the request. Usernames in the format „participant@domain.com“ will be transformed from the beginning of the value to the „@“-sign.

2.HTTP(s)/WebDAV Interfaces to SLOX

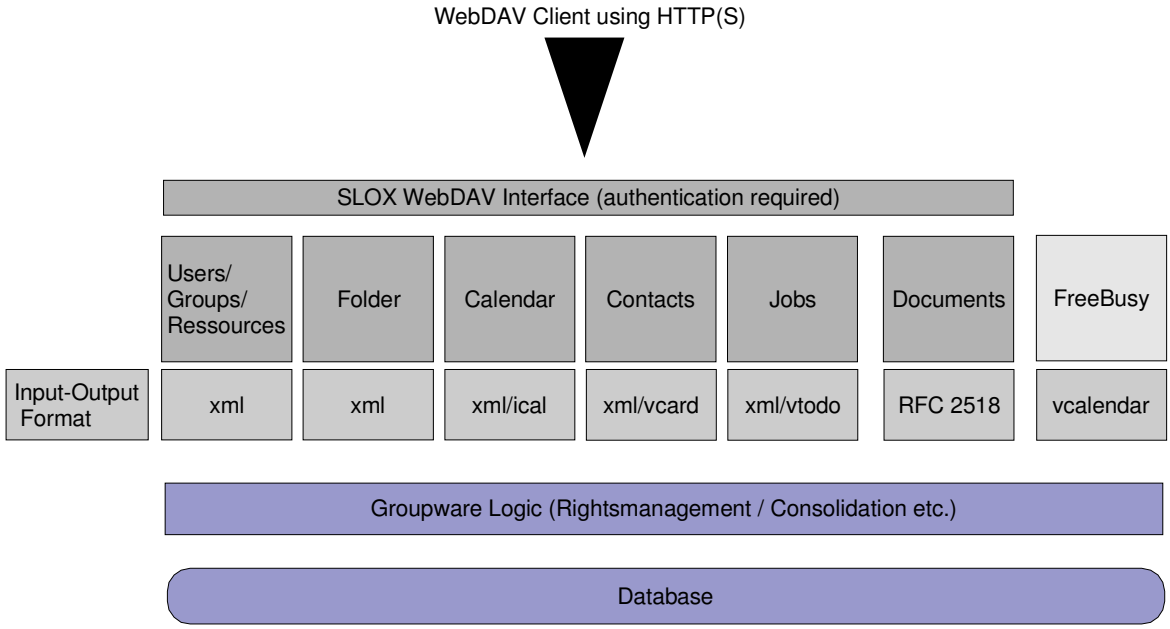
SLOX provides a set of interfaces for the following modules:

- Users/Groups and Resources
- FreeBusy Informations (Calendar)
- SLOX folders

- Appointments
- Tasks
- Contacts

- Documents

This interfaces was designed for third party products or services. We have written sample applications to show you some of the possibilities.
BTW: Our own solutions (like the Netline Palm Sync or even the iSLOX) are based on this interface.



3.Find Users/Groups/Resources

With the SLOX WebDAV Interface you can request users, groups and resources created on SLOX. The request supports search methods for users, groups and resources. The search method supports wildcards.
Creating, changing and any modification is not possible.

To return all informations as XML just send this request:
http://[SLOX_IP]/servlet/webdav.groupuser?user=*&group=*&groupres=*&res=*&details=t

The parameter „details=t“ returns more informations about the groups and resource-groups. With this parameter, the groups are shown expanded with the members or for resources-groups with the included resources.

Here is an example response:

```
<groupuser>
 <pattern>
  <group>*</group>
  <userres>*</userres>
  <groupres>*</groupres>
 </pattern>

 <group>
 <uid>group1</uid>
 <member>test004</member>
 <member>test003</member>
 </group>

 <user>
  <uid>test004</uid>
  <member>group1</member>
  <mail>test004@netline.de</mail>
  <forename/>
  <surename>test004</surename>
  </user>
  <user>
  <uid>test003</uid>
  <member>group1</member>
  <mail>test003@netline.de</mail>
  <forename/>
  <surename>test003</surename>
 </user>

 <userres>
 <uid>beamer</uid>
 </userres>

 <groupres>
 <uid>misc</uid>
 </groupres>
</groupuser>
```

4.FreeBusy interface

The SLOX FreeBusy Interface is the only interface without necessary authentification. With this interface, the clients can request all dates/times for an user, a group or a resource.

Here is a good point to say, that all dates and times should be submitted as Unix Long (a Unix Long starts at January 1, 1970, 0:00:00 GMT and it specifies the number of milliseconds since this date). All dates and times, that the client receives from the SLOX WebDAV Interface have this format too.

In order to request this information, just make a http(s) connection with the following URL:

http://[SLOXIP]/servlet/webdav.freebusy?username=[NAME]&server=[SERVER]&start=[long]&end=[long]

[SLOXIP] should be replaced with the name or IP-Address from SLOX
[USER] should be replaced with a valid user login located on SLOX
[SERVER] should be the base dn
[start] is an optional parameter. You can define the start date (as Unix Long) ab wann the FreeBusy informations should start
[end] is an optional parameter. This value defines the end date (as Unix Long) of the request

You receive an document, that looks like this:

```
BEGIN:VCALENDAR
VERSION:2.0
METHOD:PUBLISH
BEGIN:VFREEBUSY
ORGANIZER:[USER]@[SERVER]
FREEBUSY;FBTYPE=BUSY-TENTATIVE:20030212T100000Z/20030212T110000Z
FREEBUSY;FBTYPE=BUSY-UNAVAILABLE:20030303T080000Z/20030303T170000Z
FREEBUSY:FBTYPE=BUSY:20030417T080000Z/20030417T170000Z
END:VFREEBUSY
END:VCALENDAR
```

If you want to know more about the given format, please check out the references, that are provided at the end of this paper.

5.Informations about SLOX folders

To structure your informations, the easiest way is to use the SLOX Folder Interface. This interface gives you the facility to request the complete folder structure from SLOX, create and edit folders, move folders and delete them.
A folder can obtain rights for the folder itself and for the objects inside this folder.

SLOX support the following folder types:
- Calendar (cal)
- Tasks (task)
- Contacts (addr)
- Unbound (folder)

The „unbound“ folders are for any folder outside SLOX whatever SLOX do not support. This type was implemented for supporting client structures outside SLOX with any foldertype.
With this type it is possible to optain every folder structrure on SLOX and in any client.

Folders can obtain the following rights:
- view a folder (read rights)
- edit a folder (write rights)
- object read (read rights for contained objects)
- object write (write rights for contained objects)

When the user wants to create an object in a folder (an appointment for example), the rights of the appointment gains as minimal rights the object rights (*object read* and *object write*) from the folder. The minimal rights from objects in folders cannot be removed (neither in the webfrontend nor over the WebDAV interface).

Objects inside folders cannot be moved to another folder.

For every WebDAV request in calendar, tasks and contacts, you can specify a „folder_id“, to get all objects, that are in the specified folder.

The operations provided by this interface are listed below:

- create a folder
- editing a folder
- change a folders object type from unbound to anything
- move a folder
- delete a folder
- get the complete foldertree
- get new folders
- get updated folders
- get new and updated folders
- get new and updated folders (complete structure from the root level to the new/updated folder)
- get all id's of updated folders
- get deleted folders

The usage and a detailed description for the WebDAV interface of these operations are listed in the following sections. All the XML „get“ examples are based on the following

foldertree (the picture is taken from the webinterface).

Each folder in this foldertree has a „folder_id“. If a folder should be created in another folder, you have to specify the root folder (id of the folder where the new subfolder should be created in). If a folder should be created in the root level, the root folder should be „0“ (this indicates the root folder).

5.1.1Create a folder

The following PROPPATCH/PUT is an example to create a folder. The folder should have the following details:

- Foldername: „WebDAV Folder“
- Foldertype: Calendar (cal)
- The folder should be created in the root level (parentfolder = „0“)
- The group *users* should see the folder
- Nobody (except the creator) should be able to edit this folder
- The group *users* should be in every object-„readright“ in this folder
- The group *users* should be in every object-„writeright“ in this folder

```
PUT /servlet/webdav.folders/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:propertyupdate xmlns:D="DAV">
 <D:set>
 <D:prop>
 <S:name xmlns:S="SLOX:">WebDAV Folder</S:name>
 <S:type xmlns:S="SLOX:">cal</S:type>
 <S:parentfolder xmlns:S="SLOX:">0</S:parentfolder>
 <S:readrights xmlns:S="SLOX:">
 <S:group>users</S:group>
 </S:readrights>
 <S:writerights xmlns:S="SLOX:">
 </S:writerights>
 <S:objectreadrights xmlns:S="SLOX:">
 <S:group>users</S:group>
 </S:objectreadrights>
 <S:objectwriterights xmlns:S="SLOX:">
 <S:group>users</S:group>
 </S:objectwriterights>
 </D:prop>
 </D:set>
 </D:propertyupdate>
</D:multistatus>
```

The response of this PROPPATCH/PUT should look like the following:

```
HTTP/1.1 207 Response
Date: Mon, 20 Oct 2003 12:34:18 GMT
Server: Apache/1.3.26 (UnitedLinux) ApacheJServ/1.1.2
Set-Cookie: JServSessionIdroot=wns4oz1gl1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>1756</D:href>
    <D:propstat>
      <D:prop>
        <S:sloxid xmlns:S="SLOX:">1756</S:sloxid>
      </D:prop>
     <D:status>HTTP/1.1 200 OK</D:status>
     <D:responsedescription>OK</D:responsedescription>
   </D:propstat>
 </D:response>
</D:multistatus>
```

As you can see in the response, there is a field named „sloxid“. This „sloxid“ is a unique id of the new generated folder.

5.1.2 Editing a folder

If you want to update a folder, you just have to send the fields, that you want to edit (foldername and rights for example), and the id of the folder, that you want to change (field „sloxid“). The other fields will not be changed in the update.
For example, we want to change the foldername and add the group *users* to the write rights of the folder, which was created in 5.1.1, then the PROPPATCH/PUT request file would look like this:

```
PUT /servlet/webdav.folders/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
 <D:prop>
 <S:name xmlns:S="SLOX:">WebDAV Folder Changed</S:name>
 <S:sloxid xmlns:S="SLOX:">1756</S:sloxid>
 <S:writerights xmlns:S="SLOX:">
 <S:group>users</S:group>
 </S:writerights>
 </D:prop>
 </D:set>
</D:propertyupdate>
</D:multistatus>
```

If somebody wants to change a folder via WebDAV, the WebDAV Interface checks if the user of the request has rights to change the folder. If the user, who wants to change the folder has not enough rights, then the following response will be send:

```
HTTP/1.1 207 Response
Date: Mon, 20 Oct 2003 14:05:01 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=7l5gm33k61; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>1756</D:href>
    <D:propstat>
      <D:prop>
        <S:sloxid xmlns:S="SLOX:">1699</S:sloxid>
      </D:prop>
      <D:status>HTTP/1.1 403 Forbidden</D:status>
      <D:responsedescription>No Permission</D:responsedescription>
    </D:propstat>
  </D:response>
</D:multistatus>
```

The field „responsedescription“ describes the reason, why the user is not allowed to change the folder. For example, he has no permission (not enough rights) to the folder he wants to change.

If everything is ok and the user has enough rights to edit the folder, the following WebDAV response should be transmitted:

```
HTTP/1.1 207 Response
Date: Mon, 20 Oct 2003 13:57:54 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=n6dn6t3e91; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>1756</D:href>
    <D:propstat>
      <D:prop>
        <S:sloxid xmlns:S="SLOX:">1756</S:sloxid>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
      <D:responsedescription>OK</D:responsedescription>
    </D:propstat>
  </D:response>
</D:multistatus>
```

5.1.3 Changing a folder object type

In the webfrontend, it is not possible to „click“ on an unbound folder to see the objects in this folder. This is because no objects can be saved in an „unbound“ folder. So it is possible to change the type of an „unbound“ folder to „cal“, „task“ or „addr“ (because there are no objects, that have to be mapped), via WebDAV. But only the type of an „unbound“ folder can be changed! It is not possible to change the type of a „cal“, „task“ or „addr“ folder!

Example:
We want to change the folder type of the folder 1770 („folderid“). The folder is an „unbound“ folder and we want to change the type to *contact*. The PROPPATCH/PUT request should look like:

```
PUT /servlet/webdav.folders/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
   <S:sloxid xmlns:S="SLOX:">1770</S:sloxid>
   <S:type xmlns:S="SLOX:">task</S:type>
  </D:prop>
 </D:set>
</D:propertyupdate>
</D:multistatus>
```

If everything works fine (the server checks if the type of the folder is really „unbound“ and if the user has enough rights to edit the folder), the WebDAV interface should return:

```
HTTP/1.1 207 Response
Date: Mon, 20 Oct 2003 15:21:07 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=ps9wj95bl1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
   <D:href>1770</D:href>
   <D:propstat>
    <D:prop>
     <S:sloxid xmlns:S="SLOX:">1770</S:sloxid>
    </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
    <D:responsedescription>OK</D:responsedescription>
   </D:propstat>
  </D:response>
</D:multistatus>
```

5.1.4Moving a folder

It is really simple to move a folder. You just have to know two things:
- the folder id („sloxid“) of the folder you want to move
- the folder id of the new parentfolder of the folder you want to move

All these three fields are required. A folder PROPPATCH/PUT request will look like this:

```
PUT /servlet/webdav.folders/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
   <S:sloxid xmlns:S="SLOX:">1770</S:sloxid>
   <S:parentfolder xmlns:S="SLOX:">1677</S:parentfolder>
  </D:prop>
 </D:set>
</D:propertyupdate>
</D:multistatus>
```

The WebDAV Interface will also check if the user, who wants to move the folder, has enough rights to move the folder. If an error occures, the WebDAV Interface will send a response like the one in 5.2.
When everything is ok, the response will look like:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 09:57:43 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=yayl7sv631; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
   <D:href>1770</D:href>
   <D:propstat>
    <D:prop>
     <S:sloxid xmlns:S="SLOX:">1770</S:sloxid>
    </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
    <D:responsedescription>OK</D:responsedescription>
   </D:propstat>
  </D:response>
</D:multistatus>
```

5.1.5Deleting a folder

If you want to delete a folder, you just have to know the folder id of the folder you want to delete. The WebDAV Interface will do the following operations:

- check if the user has enough rights to delete the folder
- check if the user has enough rights to delete the subfolders of the folder
- delete the given folder
- delete all subfolders of the given folders
- delete all objects, that are in the folder or contained subfolders, where the user has enough rights to delete the objects
- the objects in the folder and the subfolders, where the user has not enough rights on, the WebDAV Interface will set the folder link to „null“, so that these objects are not longer in the folder

As you see, the WebDAV Interface will do everything, that is necessary to delete a folder. As you already know, you just have to send the folder id of the folder you want to delete and a PROPPATCH/PUT request will look like the following example:

```
PUT /servlet/webdav.folders/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
   <S:sloxid xmlns:S="SLOX:">1817</S:sloxid>
  </D:prop>
 </D:set>
 <D:remove>
  <D:prop>
   <S:sloxid xmlns:S="SLOX:"></S:sloxid>
  </D:prop>
 </D:remove>
</D:propertyupdate>
```

As we said, the WebDAV Interface will now check if you are allowed to delete the folder and its subfolders. If the user can delete the folder (and its subfolders), the response will look like:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
   <D:href>1817</D:href>
   <D:propstat>
    <D:prop>
     <S:sloxid xmlns:S="SLOX:">1817</S:sloxid>
    </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
    <D:responsedescription>OK</D:responsedescription>
   </D:propstat>
  </D:response>
</D:multistatus>
```

Please take note, that you will not get further information (in this response) about the deleted and removed objects.

5.2.Foldertree update cycle

There are two logical ways to get the foldertree (PROPFIND). In first way, there is a method to get the whole foldertree in one request. This is most important for the initial getting of the folder data. The usage and the format of this method is described in 5.2.1. The second way is to use a bunch of methods, to get the new, updated, moved and deleted folders, since a specified time. This is important if you have already made a PROPFIND to get the whole foldertree and now you want to get the new, updated, moved and deleted folders. Of course, it is possible to get these updated folders with the first method (get complete foldertree). But this will cause more load on the server and on the client, because more data will be transmitted. The WebDAV PROPFIND methods, that are included in this bunch are:

- DELETE (5.2.7)
- GETALLUPDATEDIDS (5.2.2)
- NEWUPDATE (5.2.3)
- STRUCTURE (5.2.6)

To understand why you need four WebDAV methods to get new, updated, moved and deleted folders, we have to explain the right's problem and the folder NABV functionallity. First of all, the right's problem:

Like every object in SLOX, folders have also read and write rights. You can define these rights to specify, which users/groups can read the folder or which users/groups can edit it. This will cause trouble for the WebDAV Interface. Example: *User admin* creates a folder in the web interface with read rights for all groups and persons and write rights only for the *user admin*. Now an external client receives the complete foldertree from the WebDAV Interface for the *user test*. The *user test* is allowed to read the folder, which the *user admin* has created. So the *user test* gets the folder in the response of the WebDAV Interface. After that, the *user admin* changes the right of the folder, he has created, to private (only the *user admin* can see this folder). If the external client wants to get the updated folders, with the WebDAV method NEWUPDATE, the folder which the *user admin* has changed will not appear, because *user test* has no longer read rights. And that is the problem. The external client receives a response from the WebDAV Interface, where the folder is not shown as updated folder, so the client has still this folder in its list. To fix this problem we created a small workaround:
- WebDAV method GETALLUPDATEDIDS will return ALL „sloxid“s of updated folders since a specified time (also the updated folders, where *user test* has no read right to!).
- You get all updated and new folders, where the user has read right to with NEWUPDATE
- Now the client has to compare: Let's say the folder, which *user admin* has created would have the „sloxid“ 1234. In the foldertree of the client appears folder 1234. And from the response of GETALLUPDATEDIDS, the client has also the folder 1234 marked as updated. But the „sloxid“ does not appear in NEWUPDATE. OK! Now the client knows that folder 1234 has been updated, but after the update the *user test* does not longer have read right to the folder.

The NABV functionallity:
It is possible, that you get a folder in the foldertree from the WebDAV Interface, where the user has no read and no write right to. These folders are called NABV folder (Not Accessable But Visible). These folders should be visible because anywhere inside these folders is a subfolder, where the user has read rights to. And the user should be able to see ALL of the folders, where he has read right to. Example, based on foldertree example (picture 1):
The *user admin* has created an „unbound“ folder „Projects“ in root level with read rights only for *user admin*, and the „unbound“ subfolder „Project WebDAV Interface“ with read rights for all groups and persons. Of course, an external client should be able to display the folder „Project WebDAV Interface“, but therefore the client has also to display folder „Projects“. So folder „Projects“ should be „NABV“. To identify a NABV folder in the WebDAV Interface response, there is a flag (called „accessright“) which is true (if folder is accessable, and not NABV) or false (it is a NABV folder).
The NABV folders are important for the WebDAV command STRUCTURE.
For example:
User admin creates folder „Projects“ in root level with read rights only for *user admin*.

Then, an external client gets the complete foldertree for *user test*. The folder „Projects will not appear in this tree, because test has no read rights to the folder and „Projects“ has also no subfolders where *user test* has read rights to. Then *user admin* creates the folder „Project WebDAV Interface“, with read rights for all groups and users, as subfolder of „Projects“. After that the external client gets all new and updated folders with WevDAV method NEWUPDATE. In this response appears the folder „Project WebDAV Interface“. But this folder has „Projects“ as parentfolder, but this folder is not yet in the foldertree of the client. In this situation the client should run the WebDAV command STRUCTURE. The response will return a foldertree with all folders from the root level to the updated folders. You can find a detailed response example in section 5.2.6.

Now it should be understandable why you have to use the commands DELETE, GETALLUPDATEDIDS, NEWUPDATE and STRUCTURE to update a foldertree on the client.

5.2.1Get the complete foldertree

The WebDAV PROPFIND command ALL on folders will return a structured foldertree with all folders where the user is allowed to read. NABV folders are also included. The root folder is marked as tag <rootfolder>. The structure will look like the following example (without folder data tags):

```
<rootfolder>
 <folder>
  <name>Folder 1</name>
  <folder>
   <name>Subfolder 1 of Folder 1</name>
  </folder>
 </folder>
</rootfolder>
```

A PROPFIND request should look like:

```
PROPFIND /servlet/webdav.folders HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns="DAV:">
 <D:prop>
  <S:objecttype xmlns:S="SLOX:">all</S:objecttype>
 </D:prop>
</D:propfind>
```

The following response example is based on the foldertree example picture:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) ApacheJServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>rootfolder</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection />
        </D:resourcetype>
        <S:rootfolder xmlns:S="SLOX:">
          <S:folder>
            <S:name>My Private Appointments</S:name>
            <S:sloxid>1677</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>cal</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227830000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228180726</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
          </S:folder>
          <S:folder>
            <S:name>My Private Contacts</S:name>
            <S:sloxid>1676</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>addr</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227815000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228180726</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
          </S:folder>
          <S:folder>
            <S:name>My Private Jobs</S:name>
            <S:sloxid>1678</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>task</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
           <S:creationTime>1066227844000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228180726</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
          </S:folder>
          <S:folder>
            <S:name>Projects</S:name>
            <S:sloxid>1679</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>folder</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227859000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
```

<S:lastModified>1066228180731</S:lastModified>
<S:folderreadrights />
<S:folderwriterights />
<S:objectreadrights />
<S:objectwriterights />
<S:folder>
<S:name>Project Outlook Interface ISLOX</S:name>
<S:sloxid>1683</S:sloxid>
<S:parentid>1679</S:parentid>
<S:type>folder</S:type>
<S:owner>benni</S:owner>
<S:createFrom>Benjamin Otterbach</S:createFrom>
<S:creationTime>1066227961000</S:creationTime>
<S:accessright>true</S:accessright>
<S:changeFrom>Benjamin Otterbach</S:changeFrom>
<S:lastModified>1066228180726</S:lastModified>
<S:folderreadrights />
<S:folderwriterights />
<S:objectreadrights />
<S:objectwriterights />
<S:folder>
<S:name>Appointments</S:name>
<S:sloxid>1685</S:sloxid>
<S:parentid>1683</S:parentid>
<S:type>cal</S:type>
<S:owner>benni</S:owner>
<S:createFrom>Benjamin Otterbach</S:createFrom>
<S:creationTime>1066227996000</S:creationTime>
<S:accessright>true</S:accessright>
<S:changeFrom>Benjamin Otterbach</S:changeFrom>
<S:lastModified>1066228180726</S:lastModified>
<S:folderreadrights />
<S:folderwriterights />
<S:objectreadrights>
<S:group>entwicklung</S:group>
<S:group>users</S:group>
</S:objectreadrights>
<S:objectwriterights />
</S:folder>
<S:folder>
<S:name>Contacts</S:name>
<S:sloxid>1684</S:sloxid>
<S:parentid>1683</S:parentid>
<S:type>addr</S:type>
<S:owner>benni</S:owner>
<S:createFrom>Benjamin Otterbach</S:createFrom>
<S:creationTime>1066227976000</S:creationTime>
<S:accessright>true</S:accessright>
<S:changeFrom>Benjamin Otterbach</S:changeFrom>
<S:lastModified>1066228180726</S:lastModified>
<S:folderreadrights />
<S:folderwriterights />
<S:objectreadrights>
<S:group>entwicklung</S:group>
</S:objectreadrights>
<S:objectwriterights />
</S:folder>
<S:folder>
<S:name>Jobs</S:name>
<S:sloxid>1686</S:sloxid>
<S:parentid>1683</S:parentid>
<S:type>task</S:type>
<S:owner>benni</S:owner>
<S:createFrom>Benjamin Otterbach</S:createFrom>
<S:creationTime>1066228008000</S:creationTime>
<S:accessright>true</S:accessright>
<S:changeFrom>Benjamin Otterbach</S:changeFrom>
<S:lastModified>1066228180726</S:lastModified>
<S:folderreadrights />
<S:folderwriterights />
<S:objectreadrights>
<S:group>entwicklung</S:group>
</S:objectreadrights>
<S:objectwriterights />
</S:folder>
</S:folder>
<S:folder>
<S:name>Project WebDAV Interface</S:name>

```
          <S:sloxid>1680</S:sloxid>
          <S:parentid>1679</S:parentid>
          <S:type>folder</S:type>
          <S:owner>benni</S:owner>
          <S:createFrom>Benjamin Otterbach</S:createFrom>
          <S:creationTime>1066227883000</S:creationTime>
          <S:accessright>true</S:accessright>
          <S:changeFrom>Benjamin Otterbach</S:changeFrom>
          <S:lastModified>1066228180731</S:lastModified>
          <S:folderreadrights>
            <S:group>entwicklung</S:group>
          </S:folderreadrights>
          <S:folderwriterights />
          <S:objectreadrights />
          <S:objectwriterights />
          <S:folder>
            <S:name>Jobs</S:name>
            <S:sloxid>1682</S:sloxid>
            <S:parentid>1680</S:parentid>
            <S:type>task</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227937000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228180726</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights>
              <S:group>entwicklung</S:group>
            </S:objectreadrights>
            <S:objectwriterights />
          </S:folder>
          <S:folder>
            <S:name>Meetings</S:name>
            <S:sloxid>1681</S:sloxid>
            <S:parentid>1680</S:parentid>
            <S:type>cal</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227919000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228180726</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights>
              <S:group>entwicklung</S:group>
            </S:objectreadrights>
            <S:objectwriterights />
          </S:folder>
        </S:folder>
      </S:folder>
    </S:rootfolder>
  </D:prop>
  <D:status>HTTP/1.1 200 OK</D:status>
 </D:propstat>
</D:response>
</D:multistatus>
```

Tag description:
<type> : The type of the folder (cal = calendar, task = task, addr = contact, folder = unbound)
<folderreadrights> : read rights of the folder
<folderwriterights> : write rights of the folder
<objectreadrights> : minimal read rights of the objects in the folder
<objectwriterights> : minimal write rights of the objects in the folder
<accessright> : false if the folder is a NABV folder

All fields with dates (creationTime, lastModified) will be returned as Unix Long.

5.2.2Get the ID's of all updated folder since a specified date/time

An external WebDAV client can determine all updated folder id's with the WebDAV command GETALLUPDATEDIDS. This method is required for the in 5.2 described update handling. A request should look like the following example:

```
PROPFIND /servlet/webdav.calendar HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns="DAV:">
 <D:prop>
   <S:objecttype xmlns:S="SLOX:">getallupdatedids</S:objecttype>
   <S:startdate xmlns:S="SLOX:">1066228181726 </S:startdate>
 </D:prop>
</D:propfind>
```

The response of this request will contain a list of ALL updated folder id's which were changed since the specified date/time. The response could look like:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) ApacheJServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
  <D:href>rootfolder</D:href>
  <D:propstat>
    <D:prop>
      <D:resourcetype>
        <D:collection />
      </D:resourcetype>
      <S:rootfolder xmlns:S="SLOX:">
        <S:folder>
          <S:sloxid>1683</S:sloxid>
        </S:folder>
      </S:rootfolder>
    </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
   </D:propstat>
  </D:response>
</D:multistatus>
```

This response shows, that folder 1683 has been changed since the specified date/time.

5.2.3Get new and updated folders since a specified date/time

An external WebDAV client can determine new and updated folders with the WebDAV command NEWUPDATE. This method will return a list of folders, which were created or updated since the specified date/time and the user has read right to. This method is required for the in 5.2 described update handling. A request should look like the following example:

```
PROPFIND /servlet/webdav.folders HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns="DAV:">
 <D:prop>
  <S:objecttype xmlns:S="SLOX:">newupdate</S:objecttype>
  <S:startdate xmlns:S="SLOX:">1066228181726 </S:startdate>
 </D:prop>
</D:propfind>
```

The response would look like:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>rootfolder</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection />
        </D:resourcetype>
        <S:rootfolder xmlns:S="SLOX:">
          <S:folder>
            <S:name>Project Outlook Interface ISLOX</S:name>
            <S:sloxid>1683</S:sloxid>
            <S:parentid>1679</S:parentid>
            <S:type>folder</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227961000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228315000</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
          </S:folder>
        </S:rootfolder>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

5.2.4Get new folders since a specified date/time

You can also split the WebDAV command NEW UPDATE in NEW and UPDATE. If you run the WebDAV PROPFIND command NEW, you will get the new folders which the user has read right to, since the specified time. The request is the same like the one of NEWUPDATE (5.2.3), just change the „objecttype“ to:

```
<S:objecttype xmlns:S="SLOX:">new</S:objecttype>
```

The response format is the same like in NEWUPDATE (5.2.3).

5.2.5Get updated folders since a specified date/time

You can also split the WebDAV command NEW UPDATE in NEW and UPDATE. If you run the WebDAV PROPFIND command UPDATE, you will get the updated folders, which the user has read right to, since the specified time. The request is the same like the one of NEWUPDATE (5.2.3), just change the „objecttype“ to:

```
<S:objecttype xmlns:S="SLOX:">update</S:objecttype>
```

The response format is the same like in NEWUPDATE (5.2.3).

5.2.6Get the folder structure from rootfolder to new/updated folders since a specified time

This method is required for the in 5.2 described update handling. With the WebDAV PROPFIND command STRUCTURE, you will get a list of the new and updated folders. But this method will give you further information. You will also get the complete foldertree from rootfolder to the new or updated folder. For example, folder „Meetings“ (Root/Projects/Project WebDAV Interface) has been updated and if you run the WebDAV STRUCTURE command, than you will get this structure (without detailed folderdata):

```
<rootfolder>
  <folder>
    <name>Projects</name>
    <folder>
      <name>Project WebDAV Interface</name>
      <folder>
        <name>Meetings</name>
      </folder>
    </folder>
  </folder>
</rootfolder>
```

The last folderitem in the structure is allways the new/updated folder!

A request example, where the folders („sloxid“) 1680 and 1683 have been changed since the given date/time:

```
PROPFIND /servlet/webdav.folders HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns="DAV:">
 <D:prop>
 <S:objecttype xmlns:S="SLOX:">structure</S:objecttype>
 <S:startdate xmlns:S="SLOX:">1066228181726</S:startdate>
 </D:prop>
</D:propfind>
```

The response would look like:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) ApacheJServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"


<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>rootfolder</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection />
        </D:resourcetype>
        <S:rootfolder xmlns:S="SLOX:">
          <S:folder>
            <S:name>Projects</S:name>
            <S:sloxid>1679</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>folder</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227859000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228518202</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
            <S:folder>
              <S:name>Project WebDAV Interface</S:name>
              <S:sloxid>1680</S:sloxid>
              <S:parentid>1679</S:parentid>
              <S:type>folder</S:type>
              <S:owner>benni</S:owner>
              <S:createFrom>Benjamin Otterbach</S:createFrom>
              <S:creationTime>1066227883000</S:creationTime>
              <S:accessright>true</S:accessright>
              <S:changeFrom>Benjamin Otterbach</S:changeFrom>
              <S:lastModified>1066228518055</S:lastModified>
              <S:folderreadrights>
                <S:group>entwicklung</S:group>
              </S:folderreadrights>
              <S:folderwriterights />
              <S:objectreadrights />
              <S:objectwriterights />
            </S:folder>
          </S:folder>
          <S:folder>
            <S:name>Projects</S:name>
            <S:sloxid>1679</S:sloxid>
            <S:parentid>0</S:parentid>
            <S:type>folder</S:type>
            <S:owner>benni</S:owner>
            <S:createFrom>Benjamin Otterbach</S:createFrom>
            <S:creationTime>1066227859000</S:creationTime>
            <S:accessright>true</S:accessright>
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228518442</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
            <S:folder>
              <S:name>Project Outlook Interface ISLOX</S:name>
              <S:sloxid>1683</S:sloxid>
              <S:parentid>1679</S:parentid>
              <S:type>folder</S:type>
              <S:owner>benni</S:owner>
              <S:createFrom>Benjamin Otterbach</S:createFrom>
              <S:creationTime>1066227961000</S:creationTime>
              <S:accessright>true</S:accessright>
```

```
            <S:changeFrom>Benjamin Otterbach</S:changeFrom>
            <S:lastModified>1066228315000</S:lastModified>
            <S:folderreadrights />
            <S:folderwriterights />
            <S:objectreadrights />
            <S:objectwriterights />
           </S:folder>
         </S:folder>
       </S:rootfolder>
     </D:prop>
     <D:status>HTTP/1.1 200 OK</D:status>
   </D:propstat>
  </D:response>
</D:multistatus>
```

5.2.7Get ALL deleted folders since a specified date/time

With the WebDAV PROPFIND method DELETED, you can determine which folders were deleted since a specified date/time. This method is required for the in 5.2 described update handling. In the response of this method, you will find a list of the „sloxid"s of the folder, which were deleted since the specified time. Example:

```
PROPFIND /servlet/webdav.folders HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns="DAV:">
 <D:prop>
  <S:objecttype xmlns:S="SLOX:">delete</S:objecttype>
  <S:startdate xmlns:S="SLOX:">1066228181726</S:startdate>
 </D:prop>
</D:propfind>
```

Response example:

```
HTTP/1.1 207 Response
Date: Tue, 21 Oct 2003 10:29:54 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=db001tvwx1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>rootfolder</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection />
        </D:resourcetype>
        <S:rootfolder xmlns:S="SLOX:">
          <S:folder>
            <S:sloxid>1675</S:sloxid>
          </S:folder>
          <S:folder>
            <S:sloxid>1672</S:sloxid>
          </S:folder>
          <S:folder>
            <S:sloxid>1668</S:sloxid>
          </S:folder>
        </S:rootfolder>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

In this response example, you can see that folders 1675, 1672 and 1668 were deleted since the given date/time from the request.

6.SLOX Appointment handling

The SLOX WebDAV Interface for appointments provides a set of functions to manage all kind of appointments. The most functionallity of the webinterface is also available with the SLOX WebDAV Interface.

The provided operations by this interface are listed below:

- creating and editing single and group appointments
- deleting appointments
- conflict requests
- setting and changing participants, read-/ and write rights
- notify members via mail if an appointment has been changed or created
- setting reminder for mail notifications
- automatic hard conflict detection (it is not possible to overbook resources)
- automatic mail notifications to participants if an appointment was deleted
- automatic mail notification to participants if a participant was removed
- bind appointments to a folder
- creating and deleting recurrent appointments

Of course, it is possible to request all appointments for a specific user or even a single appointment if the unique id („sloxid“) is known. If a folder is given in the request, then only appointments for this folder will be returned.

There are also some helpful example applications if anybody wants to develop own solutions. Take a look into the section Example Applications to find out more.

With the PROPFIND method you can request appointments from SLOX:

```
PROPFIND /servlet/webdav.calendar HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}

<?xml version="1.0" encoding="UTF-8"?>
<propfind xmlns="DAV:">
 <prop>
  <lastsync xmlns:S="SLOX:">0</lastsync>
  <folderid xmlns:S="SLOX:"></folderid>
  <objecttype xmlns:S="SLOX:">all, new, update, delete</objecttype>
 </prop>
</propfind>
```

With the field „lastsync“ you can request appointments since the given date. A valid long must be a Unix Long, that starts January 1, 1970, 00:00:00 GMT and it specifies the number of milliseconds since this date. The value 0 should response all appointments.

If a „folderid“ is given, then only appointments inside this folder will be returned.
You can let this field empty or just do not send it and then all appointments will be returned.

„objecttype“ can be combined. For example, „new, update, delete“ is the same as „all“. If you only want changed appointments, then send only the tag „update“.

If you make a request without parameter, then these default parameters will be taken for the request:

lastsync = 0
folderid =
objectteype = all

Below you find an example response of SLOX, but before some fields should be explained:

All fields with dates as value will be also returned as Unix Long (see desciption above). There is one exception. If there is a „full_time“ event, the date will be calculated in GMT, but the start time should be 00:00 in GMT. The complete timezone calculation for any date is part of the client application.

The value in the field „reminder“ is in minutes before the startdate („begins“).

Appointments supports four appointment types (field „type“):

reserved = date1
temporary = date2
absent on business = date3
holiday = date4

Every apointment has a unique id in the xml tag „sloxid“. With this value it is possible to fetch, change and delete a single appointment. This value cannot be changed.

Recurring appointments are a little bit different and you find further informations in this documentation. The fields for recurring appointments are:
sequenceid, date_sequence, deleteexceptions, changeexceptions and sequenceprop.

And here is an example response (normal appointment):

```
HTTP/1.1 207 Response
Date: Fri, 17 Oct 2003 12:33:32 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2 mod_webapp/1.2.0-dev mod_ssl/2.8.10 OpenSSL/0.9.6g mod_perl/1.27
Set-Cookie: JServSessionIdroot=o0fbg09fy1; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>200</D:href>
  <D:propstat>
   <D:prop>
    <D:getcontenttype>text/html</D:getcontenttype>
    <S:sloxstatus xmlns:S="SLOX:">CREATE</S:sloxstatus>
    <S:sequenceid xmlns:S="SLOX:" />
    <S:creationtime xmlns:S="SLOX:">1066393892000</S:creationtime>
    <S:createfrom xmlns:S="SLOX:">test001</S:createfrom>
    <S:lastmodified xmlns:S="SLOX:">1066394195000</S:lastmodified>
    <S:modifiedfrom xmlns:S="SLOX:">test001</S:modifiedfrom>
    <S:full_time xmlns:S="SLOX:"/>
    <S:owner xmlns:S="SLOX:">test001</S:owner>
    <S:deleteright xmlns:S="SLOX:" />
    <S:folderid xmlns:S="SLOX:" />
    <S:begins xmlns:S="SLOX:">1066370400000</S:begins>
    <S:ends xmlns:S="SLOX:">1066402800000</S:ends>
    <S:reminder xmlns:S="SLOX:">1440</S:reminder>
    <S:members xmlns:S="SLOX:">
     <S:member confirm="none">test002</S:member>
     <S:member confirm="none">test001</S:member>
    </S:members>
    <S:resources xmlns:S="SLOX:">
     <S:resource>konferenzraum 1</S:resource>
    </S:resources>
    <S:sloxid xmlns:S="SLOX:">200</S:sloxid>
    <S:sequence_position xmlns:S="SLOX:" />
    <S:readrights xmlns:S="SLOX:">
     <S:member>test002</S:member>
     <S:member>test001</S:member>
     <S:group>group1</S:group>
    </S:readrights>
    <S:writerights xmlns:S="SLOX:">
     <S:member>test002</S:member>
     <S:member>test001</S:member>
    </S:writerights>
    <S:title xmlns:S="SLOX:">[description]</S:title>
    <S:location xmlns:S="SLOX:">[location]</S:location>
    <S:type xmlns:S="SLOX:">date1</S:type>
    <S:description xmlns:S="SLOX:">[detail]</S:description>
    <S:date_sequence xmlns:S="SLOX:">no</S:date_sequence>
    <S:sequenceprop xmlns:S="SLOX:">ds|no|</S:sequenceprop>
    <S:deleteexceptions xmlns:S="SLOX:" />
    <S:changeexceptions xmlns:S="SLOX:" />
   </D:prop>
   <D:status>HTTP/1.1 200 OK</D:status>
   <D:responsedescription>HTTP/1.1 200 OK</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

With the PUT or PROPPATCH it is possible to create/update appointments on SLOX:

```
PROPFIND /servlet/webdav.calendar /file.xmlHTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}
```

or

```
PUT /servlet/webdav.calendar/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:propertyupdate xmlns:D="DAV">
  <D:set>
   <D:prop>
    <S:begins xmlns:S="SLOX:">1059841457362</S:begins>
    <S:ends xmlns:S="SLOX:">1060000000000</S:ends>
    <S:clientid xmlns:S="SLOX:">clientid</S:clientid>
    <S:folderid xmlns:S="SLOX:"></S:folderid>
    <S:title xmlns:S="SLOX:">WebDAV Insert Example</S:title>
    <S:description xmlns:S="SLOX:">This will create an appointment</S:description>
    <S:location xmlns:S="SLOX:">Anywhere</S:location>
    <S:type xmlns:S="SLOX:">date2</S:type>
    <S:reminder xmlns:S="SLOX:">1440</S:reminder>
    <S:mailtomembers xmlns:S="SLOX:">yes</S:mailtomembers>
   </D:prop>
  </D:set>
 </D:propertyupdate>
</D:multistatus>
```

And the response for this example insert should look like this one:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>201</D:href>
  <D:propstat>
   <D:prop>
    <S:sloxid xmlns:S="SLOX:">201</S:sloxid>
    <S:clientid xmlns:S="SLOX:">clientid</S:clientid>
    </D:prop>
   <D:status>HTTP/1.1 200 OK</D:status>
   <D:responsedescription>OK</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

As you can see, the client can define a „clientid“ in the request and this value is also in the response, including the newly generated „sloxid“. This is for an easy mapping function character.

Update this appointment is very easy:
just add the tag „sloxid“ in your PUT or PROPPATCH xml file and send it to SLOX:

```
<S:sloxid xmlns:S="SLOX:">201</S:sloxid>
```

If you want to fetch your updated appointment, make a PROPFIND with the „sloxid“ and only this item will be returned:

```
PROPFIND /servlet/webdav.calendar/[sloxid] HTTP/1.1
```

Deleting appointments with the SLOX WebDAV Interface:

If you want to delete an appointment you should know the „sloxid“. The delete command looks like this:

```
DELETE  /servlet/webdav.calendar/[SLOXID] HTTP/1.1
Content-Type: text/xml
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded]
```

or by sending a PUT/PROPPATCH with this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
    <S:clientid xmlns:S="SLOX:">[CLIENTID]</S:clientid>
  </D:prop>
 </D:set>
 <D:remove>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:"></S:sloxid>
  </D:prop>
 </D:remove>
</D:propertyupdate>
</D:multistatus>
```

Please take note, that you have to specify the „sloxid“ after the url to delete this appointment.
If you do not have the rights to delete the appointment, you will get an error message, that you are not allowed to delete this appointment. An example error message is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>201</D:href>
  <D:propstat>
   <D:prop>
    <S:sloxid xmlns:S="SLOX:">201</S:sloxid>
   </D:prop>
   <D:status>HTTP/1.1 403 Forbidden</D:status>
   <D:responsedescription>No Permission</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

The field „responsedescription“ can be used to inform the client about the kind of the error.

Specials inside the SLOX WebDAV Interface for appointments:

The client has the facility to make a conflict request for an insert or update. For that just make a normal insert/update request and add the xml tag

```
<S:appointment_request xmlns:S="SLOX:">yes</S:appointment_request>
```

in your xml request. The result for the given times and participants/resources is a xml file with all conflicts, which looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>[SLOXID]</D:href>
  <D:propstat>
   <D:prop>
    <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
   </D:prop>
   <D:status>HTTP/1.1 200 OK</D:status>
   <D:responsedescription>[CONFLICT LEVEL]</D:responsedescription>
  </D:propstat>
 </D:response>
 <D:response>
  <D:href>[SLOXID]</D:href>
  <D:propstat>
  <D:prop>
   <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
  </D:prop>
  <D:status>HTTP/1.1 200 OK</D:status>
  <D:responsedescription>[CONFLICT LEVEL]</D:responsedescription>
 </D:propstat>
</D:response>
</D:multistatus>
```

Inside the results are the conflicting „sloxid“s and the conflict level of all the conflicts. SLOX supports three different conflict levels:

Soft Conflict = can be ignored (user can create/change the appointment)
Hard Conflict = not ignorable (it is not possible to create/change this appointment)
Holiday Conflict = not ignorable (it is not possible to create/change this appointment)

If the developer of the client wants to show more informations, he can request now each single appointment with the „sloxid“ of the response to show a more detailed informations about the conflict.

Holiday Conflicts cannot be requested!

Setting the delete right for an appointment:

With the SLOX WebDAV Interface you can define an owner for an object. This owner can access to the objects, also when he is not listed in the read/write permissions. And only this owner can delete the object!
The owner tag can be identified by the following tag:

```
<S:deleteRight xmlns:S="SLOX:">[owner of object (uid)]</S:deleteRight>
```

Recurring appointments:

Each sequence returns only one object. Inside this object you find the deleted objects (by position, csv) in field "deleteexceptions". Changed objects can be found in field (csv) "changeexceptions".
Just request this changed object(s) with a propfind:

PROPFIND /servlet/webdav.calendar/[changeexceptionID] HTTP/1.1

The field "sequence_position" indicates the position inside the sequence.
The field "sequence_date_position" always contains the original date of the appointment, even if the appointment was moved.
Delete and change single sequences (exceptions) can be done by the tag adding "sequence_position" into the proppatch or by adding the field "sequence_date_position". Then just send the normal proppatch with the „sloxid“ of the "MAIN" sequence object!

To change the whole sequence, just update the given object ("MAIN") and let the field "sequence_position" and the "sequence_date_postion" empty. In this case, the "changeexceptions" will be nulled!

Fields for a daily sequence:

```
<S:date_sequence xmlns:S="SLOX:">daily</S:date_sequence>
<S:daily_value xmlns:S="SLOX:">[daily interval]</S:daily_value>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

Fields for a weekly sequence:

```
<S:date_sequence xmlns:S="SLOX:">daily</S:date_sequence>
<S:daily_value xmlns:S="SLOX:">[daily interval]</S:daily_value>
<S:weekly_value xmlns:S="SLOX:">[weekly interval]</S:weekly_value>
<S:weekly_day_1 xmlns:S="SLOX:">[1 (Sunday)]</S:weekly_day_1>
<S:weekly_day_2 xmlns:S="SLOX:">[2 (Monday)]</S:weekly_day_2>
<S:weekly_day_3 xmlns:S="SLOX:">[3 (Tuesday)]</S:weekly_day_3>
<S:weekly_day_4 xmlns:S="SLOX:">[4 (Wednesday)]</S:weekly_day_4>
<S:weekly_day_5 xmlns:S="SLOX:">[5 (Thursday)]</S:weekly_day_5>
<S:weekly_day_6 xmlns:S="SLOX:">[6 (Friday)]</S:weekly_day_6>
<S:weekly_day_7 xmlns:S="SLOX:">[7 (Saturday)]</S:weekly_day_7>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

Send only those days for a weekly sequence where SLOX should add the appointment!
Here a small example for a weekly reccuring appointment every Monday and Friday:

```
<S:date_sequence xmlns:S="SLOX:">weekly</S:date_sequence>
<S:daily_value xmlns:S="SLOX:">1</S:daily_value>
<S:weekly_day_2 xmlns:S="SLOX:">2</S:weekly_day_2>
<S:weekly_day_6 xmlns:S="SLOX:">6</S:weekly_day_6>
<S:ds_ends xmlns:S="SLOX:">1062633600000</S:ds_ends>
```

Fields for a monthly sequence:

```
<S:date_sequence xmlns:S="SLOX:">monthly</S:date_sequence>
<S:monthly_value_day xmlns:S="SLOX:">[daily interval]</S:monthly_value_day>
<S:monthly_value_month xmlns:S="SLOX:">[monthly interval]</S:monthly_value_month>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

Fields for yearly sequence:

```
<S:date_sequence xmlns:S="SLOX:">yearly</S:date_sequence>
<S:yearly_value_day xmlns:S="SLOX:">[0-31 (day of month)</S:yearly_value_day>
<S:yearly_month xmlns:S="SLOX:">[0-11 month of year]</S:yearly_month>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

For the monthly and the yearly recurring appointments exists two more types:

Monthly2:

```
<S:date_sequence xmlns:S="SLOX:">monthly2</S:date_sequence>
<S:monthly2_reccurency xmlns:S="SLOX:">[ monthly2_reccurency }</S:monthly2_reccurency>
<S:monthly2_day xmlns:S="SLOX:">[ monthly2_day ]</S:monthly2_day>
<S:monthly2_value_month xmlns:S="SLOX:">[monthly interval]</S:monthly2_value_month>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

Possible values for „monthly2_reccurency“ are:
1-4 = week of month
-1 = LAST (in combination with the field „monthly2_day“

Possible values for „monthly2_day“ are:
1-7 = for a week day (0 = Sunday, 7 = Saturday)
-1 = DAY (in combination with „monthly2_reccurency“ = -1 this means the last day in the month)
-2 = Weekday (in combination with „monthly2_reccurency“ = 2 this would be the 2. weekday in a month)
-3 = Weekend day (in combination with „monthly2_reccurency“ = 1 this would be the 1. weekend day in the month)

You can combine each option together and you have a powerful interface for monthly recurrencies.

Yearly2:

```
<S:date_sequence xmlns:S="SLOX:">yearly2</S:date_sequence>
<S:yearly2_reccurency xmlns:S="SLOX:">[yearly2_reccurency]</S:yearly2_reccurency>
<S:yearly2_day xmlns:S="SLOX:">[yearly2_day]</S:yearly2_day>
<S:yearly2_month xmlns:S="SLOX:">[month of year] (0 = januar 11 = december)</S:yearly2_month>
<S:ds_ends xmlns:S="SLOX:">[long value]</S:ds_ends>
```

Possible values for „yearly2_reccurency“ are:
1-4 = week of month
-1 = LAST

Possible values for „yearly2_day“ are:
1-7 = for a week day (0 = Sunday, 7 = Saturday)
-1 = Day
-2 = Weekday
-3 = Weekend day

Here is the same handling. With both values („yearly2_reccurency“ and „yearly2_day“ you can make reccurency appointments like:
every 2. weekday in October.

7.SLOX Tasks handling

With the SLOX WebDAV Interface you have a set of functions to manage tasks. The interface allows you to use the most of the functionalities available in the webfrontend like:

- creating private tasks or, by adding participants to the xml structure, group tasks
- deleting taskss
- setting the read-, write- or delete/owner rights
- notify participants when a job is created/updated
- setting a reminding time so the system sends you a mail at the specified time
- setting the approvement status if a user has accepted or declined a job
- attaching tasks to a folder (for details of folder, please take a look to section 5)
- fetching available tasks by single id or selection all tasks, new or updated tasks since specified time

With the PROPFIND method you can request tasks from SLOX:

```
PROPFIND /servlet/webdav.tasks HTTP/1.1

Accept-Language: de, en-us;q=0.2
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded]

<?xml version="1.0" encoding="UTF-8"?>
<propfind xmlns="DAV:">
 <prop>
  <lastsync>[longvalue]</lastsync>
  <folderid>[folderid]</folderid>
  <objecttype xmlns:S="SLOX:">[possible values: ALL, NEW, UPDATE, DELETE]</objecttype>
 </prop>
</propfind>
```

Or get a single object by id:

```
PROPFIND /servlet/webdav.tasks/[sloxid] HTTP/1.1
```

With the field „lastsync“ you can request tasks since the given date. A valid long must be a Unix Long starts at January 1, 1970, 00:00:00 GMT and it specifies the number of milliseconds since this date. The value 0 should response all tasks.

If a „folderid“ is given, then only tasks inside this folder will be returned.
You can leave this field empty or just don't send it for all tasks.

„objecttype“ can be combined. For example „new, update, delete“ is the same as „all“. If you only want changed tasks then send only the tag „update“.

If you only make a request without parameter then the default parameters will be taken for the request:

lastsync = 0
folderid =
objecttype = all

Below you find an example response of SLOX, but before some fields should be explained:

All fields with dates in it will be returnd also as Unix Long (see desciption above).

The value in the field „reminder“ is in minutes before the startdate.

Every task has a unique id in the xml tag „sloxid“. With this value it is possible to fetch a single task, change a task and delete a task. This value cannot be changed.

And here is an example response (normal job):

```
HTTP/1.1 207 Response
Date: Wed, 15 Oct 2003 13:34:56 GMT
Server: Apache/1.3.26 (UnitedLinux) Apache JServ/1.1.2
Set-Cookie: JServSessionIdroot=example; path=/
Connection: close
Content-Type: text/xml; charset="utf-8"

<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>[slox object id]</D:href>
  <D:propstat>
   <D:prop>
    <D:getcontenttype>text/html</D:getcontenttype>
    <S:sloxstatus xmlns:S="SLOX:">[possible values: CREATE | DELETE]</S:sloxstatus>
    <S:creationtime xmlns:S="SLOX:">[timestamp as long]</S:creationtime>
    <S:createfrom xmlns:S="SLOX:">[username]</S:createfrom>
    <S:lastmodified xmlns:S="SLOX:">[timestamp as long]</S:lastmodified>
    <S:modifiedfrom xmlns:S="SLOX:"></S:modifiedfrom>
    <S:owner xmlns:S="SLOX:">[username]</S:owner>
    <S:sid xmlns:S="SLOX:" />
    <S:deleteright xmlns:S="SLOX:">[username]</S:deleteright>
    <S:folderid xmlns:S="SLOX:" />
    <S:startdate xmlns:S="SLOX:">[timestamp as long]</S:startdate>
    <S:deadline xmlns:S="SLOX:">[timestamp as long]</S:deadline>
    <S:follow_up xmlns:S="SLOX:">[timestamp as long]</S:follow_up>
    <S:reminder xmlns:S="SLOX:">[reminding time as int]</S:reminder>
    <S:members xmlns:S="SLOX:">
     <S:member confirm="none">[username]</S:member>
     <S:member confirm="decline">[username]</S:member>
     <S:member confirm="accept">[username]</S:member>
    </S:members>
    <S:sloxid xmlns:S="SLOX:">[slox object id]</S:sloxid>
    <S:should_cost xmlns:S="SLOX:">[double]</S:should_cost>
    <S:is_cost xmlns:S="SLOX:">[double]</S:is_cost>
    <S:should_duration xmlns:S="SLOX:">[double]</S:should_duration>
    <S:is_duration xmlns:S="SLOX:">[double]</S:is_duration>
    <S:readrights xmlns:S="SLOX:">
     <S:member>[username]</S:member>
     <S:group>[groupname]</S:group>
    </S:readrights>
    <S:writerights xmlns:S="SLOX:">
     <S:member>[username]</S:member>
     <S:group>[groupname]</S:group>
    </S:writerights>
    <S:title xmlns:S="SLOX:">[string]</S:title>
    <S:description xmlns:S="SLOX:">[string]</S:description>
    <S:project_id xmlns:S="SLOX:">[int]</S:project_id>
    <S:priority xmlns:S="SLOX:">[int]</S:priority>
    <S:status xmlns:S="SLOX:">[int]</S:status>
    <S:duration_type xmlns:S="SLOX:">[possible values: H | D]</S:duration_type>
    <S:cost_type xmlns:S="SLOX:">[possible values: F | V]</S:cost_type>
    <S:currency xmlns:S="SLOX:">[string]</S:currency>
   </D:prop>
   <D:status>HTTP/1.1 200 OK</D:status>
   <D:responsedescription>HTTP/1.1 200 OK</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

With the PUT or PROPPATCH it is possible to create/update tasks on SLOX:

```
PROPFIND /servlet/webdav.tasks /file.xmlHTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded]
```

or

```
PUT /servlet/webdav.tasks/file.xml HTTP/1.1
Content-Type: text/xml
Content-Length: [content length]
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:propertyupdate xmlns:D="DAV">
  <D:set>
   <D:prop>
    <S:clientid xmlns:S="SLOX:">clientid</S:clientid>
    <S:startdate xmlns:S="SLOX:">1059841457362</S:startdate>
    <S:deadline xmlns:S="SLOX:">1060000000000</S:deadline>
    <S:title xmlns:S="SLOX:">WedbDav Insert Example</S:title>
    <S:description xmlns:S="SLOX:">This will create a task</S:description>
    <S:priority xmlns:S="SLOX:">1</S:priority>
    <S:status xmlns:S="SLOX:">30</S:status>
    <S:reminder xmlns:S="SLOX:">1440</S:reminder>
    <S:mailtomembers xmlns:S="SLOX:">yes</S:mailtomembers>
   </D:prop>
  </D:set>
 </D:propertyupdate>
</D:multistatus>
```

And the response for this insert example should look like this one:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>9676</D:href>
  <D:propstat>
   <D:prop>
    <S:sloxid xmlns:S="SLOX:">9676</S:sloxid>
    <S:clientid xmlns:S="SLOX:">clientid</S:clientid>
   </D:prop>
   <D:status>HTTP/1.1 200 OK</D:status>
   <D:responsedescription>OK</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

As you can see, the client can define a „clientid“ in the request and this value is also in the response including the newly generated „sloxid“. This is for easy mapping.

Update this task is very easy:
just add the tag „sloxid“ in your PUT or PROPPATCH xml file and send it to SLOX:

```
<S:sloxid xmlns:S="SLOX:">9676</S:sloxid>
```

If you want to fetch your updated task, make a PROPFIND with the „sloxid“ and only this item will be returned:

```
PROPFIND /servlet/webdav.tasks/9676 HTTP/1.1
```

Creating „private“ or „group“ tasks:

In SLOX it is possible to create „group“ or „private“ tasks. If you want to create a „group“ task, add the following tags to your xml file:

```
<S:members xmlns:S="SLOX:">
 <S:member>username (uid)</S:member>
 <S:group>users</S:group>
</S:members>
```

As you can see, you can use a mixture of single members and/or groups.
All not existing members and/or groups on SLOX would be automaticly removed by the WebDAV Interface!

Note: Members always will have „read rights“ to a task, so you do not have to set them separately (how to set read/write rights, please read section „Settting read / write rights for a task“).

If the user, who creates the task or the owner is not a member, it is automatically set to „delegated“ for this user / owner.

Setting read / write rights for a task:

As in the webfrontend, you can also define special read and/or write rights for tasks:

```
<S:readrights xmlns:S="SLOX:">
  <S:member>[username (uid)]</S:member>
  <S:group>[groupname]</S:group>
</S:readrights>

<S:writerights xmlns:S="SLOX:">
  <S:member>[username (uid)]</S:member>
  <S:group>[groupname]</S:group>
</S:writerights>
```

As you can see, you can use a mixture of single members and/or groups.
All not existing members and/or groups on SLOX would be automatically removed by the WebDAV Interface!

Note: Every user and/or group, which you give write rights to, will automatically have read rights, so that you do not have to set them separately!

Deleting tasks with the SLOX WebDAV Interface:

If you want to delete a task, you should know the „sloxid“. The delete command looks like this:

```
DELETE  /servlet/webdav.tasks/[SLOXID] HTTP/1.1
Content-Type: text/xml
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded]
```

or with send a PUT/PROPPATCH with this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
    <S:clientid xmlns:S="SLOX:">[CLIENTID]</S:clientid>
  </D:prop>
 </D:set>
 <D:remove>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:"></S:sloxid>
  </D:prop>
 </D:remove>
</D:propertyupdate>
</D:multistatus>
```

Please take note that you have to specify the „sloxid“ after the url to delete this task. If you do not have the rights to delete the task, you will get an error message that you are not allowed to delete this task. An example error message is shown below:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
 <D:response>
  <D:href>9672</D:href>
  <D:propstat>
   <D:prop>
    <S:sloxid xmlns:S="SLOX:">9672</S:sloxid>
   </D:prop>
   <D:status>HTTP/1.1 403 Forbidden</D:status>
   <D:responsedescription>No Permission</D:responsedescription>
  </D:propstat>
 </D:response>
</D:multistatus>
```

The field „responsedescription“ can be used to inform the client about the kind of the error.

Specials inside the SLOX WebDAV Interface for tasks:

Setting the delete right for a task:

With the SLOX WebDAV Interface you can define an owner of an object. This owner can access the objects where he is owner in any cases (also when he is not in the read/write permissions). And only this owner can delete the object!
The owner tag can identified by the following tag:

```
<S:deleteRight xmlns:S="SLOX:">[owner of object (uid)]</S:deleteRight>
```

Setting confirmation status for participants:

The SLOX WebDAV Interface allows you to set the confirmation status of a member. If you want to set the status you have to add the following attribute to the members tag (For details how to set members please read „Creating private or group tasks“):

```
<S:members xmlns:S="SLOX:">
 <S:member confirm="accept">username (uid)</S:member>
</S:members>
```

There are 3 different confirmation status available in SLOX:

accept = the member has accept the task

decline = the member has declined the task

none = not set

8.SLOX contact handling

The SLOX WebDAV Interface for contact management provides a workflow to manage contacts and companies. Private informations about the SLOX users are also stored in the SLOX contact management.

The contact management in SLOX consists of two main parts: first the contacts and second the companies. The SLOX WebDAV Interface combines these two parts to one contact, so that each SLOX contact provides contact informations and his attached company informations.
Contacts without an attached company only provides simple contact informations. So it is not possible to create, edit or delete a company directly. If you want to edit company informations, you must edit a contact with the attached company you want to edit. This structure allows different ways to handle the contact/company management:

- create a contact
- create a contact and create a company
- create a contact and edit a company
- edit a contact and create a company
- edit a contact and a company
- delete a contact

The SLOX WebDAV Interface does not support to delete a company. This kind of function character has a disadvantage: create two contacts with the same company will force two contacts and two companies in SLOX, but the contacts should be attached to one company. So the SLOX WebDAV Interface provides a simple method for attaching contacts. Every action with company data will check if a company with the same "name" and "postal code" exist. If a company is found, (the read rights will be checked!) then the contact will be attached to the existing company. Otherwise a new company will be created. A small overview:

- create a contact without a company
- create a contact with a new company
- create a contact attach to an existing company
- edit a contact with a new company
- edit a contact attach to an existing company
- edit a contact unbind from a company

It is impossible to edit, delete a SLOX user (internal user) through the SLOX WebDAV Interface. This can be done with the SLOX webfrontend.

Response example for a PROPFIND:

```
<?xml version="1.0" encoding="UTF-8"?>
```

<D:multistatus xmlns:D="DAV:">
 <D:response>
    <D:href>1660</D:href>
    <D:propstat>
      <D:prop>

All contact information:

<S:sloxid xmlns:S="SLOX:">1660</S:sloxid>
<S:sloxstatus xmlns:S="SLOX:">CREATE</S:sloxstatus>
<S:group_right xmlns:S="SLOX:">g</S:group_right>
<S:creationtime xmlns:S="SLOX:">1066056326000</S:creationtime>
<S:createfrom xmlns:S="SLOX:">Benjamin Frederic Pahne</S:createfrom>
<S:lastmodified xmlns:S="SLOX:">1066056507000</S:lastmodified>
<S:modifiedfrom xmlns:S="SLOX:">Benjamin Frederic Pahne</S:modifiedfrom>
<S:owner xmlns:S="SLOX:">fred</S:owner>
<S:position xmlns:S="SLOX:">Developer</S:position>
<S:salutation xmlns:S="SLOX:">Mr</S:salutation>
<S:lastname xmlns:S="SLOX:">Doe</S:lastname>
<S:firstname xmlns:S="SLOX:">John</S:firstname>
<S:title xmlns:S="SLOX:">Dr.</S:title>
<S:department xmlns:S="SLOX:">Development</S:department>
<S:birthday xmlns:S="SLOX:">1066608000000</S:birthday>
<S:phone xmlns:S="SLOX:">0123112233</S:phone>
<S:fax xmlns:S="SLOX:">0123112244</S:fax>
<S:mobile xmlns:S="SLOX:">0190112233</S:mobile>
<S:email xmlns:S="SLOX:">john@inc.org</S:email>
<S:comment xmlns:S="SLOX:">john works here since 1982</S:comment>
<S:privatemobile xmlns:S="SLOX:">0190123123</S:privatemobile>
<S:privateemail xmlns:S="SLOX:">john@private.org</S:privateemail>
<S:privatecountry xmlns:S="SLOX:">Country</S:privatecountry>
<S:privatecity xmlns:S="SLOX:">Town</S:privatecity>
<S:privatezipcode xmlns:S="SLOX:">12345</S:privatezipcode>
<S:privatestate xmlns:S="SLOX:">State</S:privatestate>
<S:privatestreet xmlns:S="SLOX:">Street</S:privatestreet>
<S:privatefax xmlns:S="SLOX:">01234112288</S:privatefax>
<S:privateurl xmlns:S="SLOX:">www.johndoe.com</S:privateurl>
<S:privatephone xmlns:S="SLOX:">01234112277</S:privatephone>
<S:phone2 xmlns:S="SLOX:">0123113311</S:phone2>
<S:fax2 xmlns:S="SLOX:">0123112255</S:fax2>
<S:mobile2 xmlns:S="SLOX:">0190112244</S:mobile2>
<S:email2 xmlns:S="SLOX:">doe@inc.org</S:email2>
<S:privatephone2 xmlns:S="SLOX:">0123123123</S:privatephone2>
<S:privatefax2 xmlns:S="SLOX:">0123123124</S:privatefax2>
<S:privatemobile2 xmlns:S="SLOX:">0190123129</S:privatemobile2>
<S:privateemail2 xmlns:S="SLOX:">john@johndoe.com</S:privateemail2>

Company informations and „addressid“:

<S:addressid xmlns:S="SLOX:">1661</S:addressid>

<S:address xmlns:S="SLOX:">
  <S:addressid>1661</S:addressid>
  <S:status>Reseller</S:status>
  <S:employees>11 - 20</S:employees>
  <S:company>Nosense Inc.</S:company>
  <S:appendix>Nosense Inc. in Ghost Town</S:appendix>
  <S:street>Street</S:street>
  <S:zipcode>12345</S:zipcode>
  <S:city>Ghost Town</S:city>
  <S:country>Germany</S:country>
  <S:state>State</S:state>
  <S:firmphone>01234123412</S:firmphone>
  <S:firmfax>01234123413</S:firmfax>
  <S:firmemail>info@nosense.org</S:firmemail>
  <S:url>www.nosense.org</S:url>
  <S:misc>hugh parking lot available</S:misc>
  <S:addresscomment>its a nice company but watch the chef</S:addresscomment>
  <S:pobox />
  <S:poboxzip />
  <S:region />
</S:address>

Folderid:

```
<S:folderid xmlns:S="SLOX:" />
```

Read-, write- and deleteright:

```
<S:readrights xmlns:S="SLOX:">
  <S:group>users</S:group>
</S:readrights>

<S:writerights xmlns:S="SLOX:">
  <S:group>users</S:group>
</S:writerights>

<S:deleteright xmlns:S="SLOX:" />
```

Insert example:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:propertyupdate>
    <D:set>
      <D:prop>
```

HINT: this tag creates the contact in a folder with the given id

```
<S:folderid xmlns:S="SLOX:">34711</S:folderid>

<S:position xmlns:S="SLOX:">Developer</S:position>
<S:salutation xmlns:S="SLOX:">Mr</S:salutation>
<S:lastname xmlns:S="SLOX:">Doe</S:lastname>
<S:firstname xmlns:S="SLOX:">John</S:firstname>
<S:title xmlns:S="SLOX:">Dr.</S:title>
<S:department xmlns:S="SLOX:">Development</S:department>
<S:birthday xmlns:S="SLOX:">1066608000000</S:birthday>
<S:phone xmlns:S="SLOX:">0123112233</S:phone>
<S:fax xmlns:S="SLOX:">0123112244</S:fax>
<S:mobile xmlns:S="SLOX:">0190112233</S:mobile>
<S:email xmlns:S="SLOX:">john@inc.org</S:email>
<S:comment xmlns:S="SLOX:">john works here since 1982</S:comment>
<S:privatemobile xmlns:S="SLOX:">0190123123</S:privatemobile>
<S:privateemail xmlns:S="SLOX:">john@private.org</S:privateemail>
<S:privatecountry xmlns:S="SLOX:">Country</S:privatecountry>
<S:privatecity xmlns:S="SLOX:">Town</S:privatecity>
<S:privatezipcode xmlns:S="SLOX:">12345</S:privatezipcode>
<S:privatestate xmlns:S="SLOX:">State</S:privatestate>
<S:privatestreet xmlns:S="SLOX:">Street</S:privatestreet>
<S:privatefax xmlns:S="SLOX:">01234112288</S:privatefax>
<S:privateurl xmlns:S="SLOX:">www.johndoe.com</S:privateurl>
<S:privatephone xmlns:S="SLOX:">01234112277</S:privatephone>
<S:phone2 xmlns:S="SLOX:">0123113311</S:phone2>
<S:fax2 xmlns:S="SLOX:">0123112255</S:fax2>
<S:mobile2 xmlns:S="SLOX:">0190112244</S:mobile2>
<S:email2 xmlns:S="SLOX:">doe@inc.org</S:email2>
<S:privatephone2 xmlns:S="SLOX:">0123123123</S:privatephone2>
<S:privatefax2 xmlns:S="SLOX:">0123123124</S:privatefax2>
<S:privatemobile2 xmlns:S="SLOX:">0190123129</S:privatemobile2>
<S:privateemail2 xmlns:S="SLOX:">john@johndoe.com</S:privateemail2>
```

HINT: Use this tag to attach the contact directly to a company

```
<S:addressid xmlns:S="SLOX:">1661</S:addressid>
```

HINT: Use this block insted of „addressid“ to create a new company or check if this company allready exists

```
<S:address xmlns:S="SLOX:">
  <S:addressid>1661</S:addressid>
  <S:status>Reseller</S:status>
  <S:employees>11 - 20</S:employees>
  <S:company>Nosense Inc.</S:company>
  <S:appendix>Nosense Inc. in Ghost Town</S:appendix>
  <S:street>Street</S:street>
  <S:zipcode>12345</S:zipcode>
  <S:city>Ghost Town</S:city>
  <S:country>Germany</S:country>
  <S:state>State</S:state>
  <S:firmphone>01234123412</S:firmphone>
  <S:firmfax>01234123413</S:firmfax>
  <S:firmemail>info@nosense.org</S:firmemail>
```

```
                    <S:url>www.nosense.org</S:url>
                    <S:misc>hugh parking lot available</S:misc>
                    <S:addresscomment>its a nice company but watch the chef</S:addresscomment>
                    <S:pobox />
                    <S:poboxzip />
                    <S:region />
                  </S:address>

                  <S:readrights xmlns:S="SLOX:">
                    <S:group>users</S:group>
                  </S:readrights>

                  <S:writerights xmlns:S="SLOX:">
                    <S:group>users</S:group>
                  </S:writerights>

                  HINT: Use deleteright to set a owner/delteright:
                  <S:deleteright xmlns:S="SLOX:" />

      </D:prop>
    </D:set>
  </D:propertyupdate>
</D:multistatus>
```

Update example:
The handling provides the same XML-tree as an insert just add a „sloxid“ to the given contact id:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:propertyupdate>
    <D:set>
      <D:prop>

                  <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
                  <S:position xmlns:S="SLOX:">Developer</S:position>
                  ...
```

Spezial arguments:
Use „all_groups_and_perons“ for full read- or writerights

```
                  <S:writerights xmlns:S="SLOX:">
                    <S:group>all_groups_and_users</S:group>
                  </S:writerights>
```

Use „folderid“ to create a contact in a folder with the given id:

```
                  <S:folderid xmlns:S="SLOX:">[FOLDERID]</S:folderid>
```

Use „addressid“ to attach a contact to an existing company with the given id:

```
                  <S:addressid xmlns:S="SLOX:">[SLOXID]</S:addressid>
```

Use a long for birthday-tag. The long should be a Unix Long.

```
                  <S:birthday xmlns:S="SLOX:">1066608000000</S:birthday>
```

Delete a contact with the SLOX WebDAV Interface:

If you want to delete a contact you should know the „sloxid“. The delete command looks like this:

```
DELETE  /servlet/webdav.contacts/[SLOXID] HTTP/1.1
Content-Type: text/xml
User-Agent: Netline WebDAV Client
Connection: Keep-Alive
Authorization: Basic [base64coded]
```

or with send a PUT/PROPPATCH with this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<D:multistatus xmlns:D="DAV:">
<D:propertyupdate xmlns:D="DAV">
 <D:set>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:">[SLOXID]</S:sloxid>
    <S:clientid xmlns:S="SLOX:">[CLIENTID]</S:clientid>
  </D:prop>
 </D:set>
 <D:remove>
  <D:prop>
    <S:sloxid xmlns:S="SLOX:"></S:sloxid>
  </D:prop>
 </D:remove>
</D:propertyupdate>
</D:multistatus>
```

9.Interface for ical, vtodo and vcards

The SLOX WebDAV Interface provides the import and export of special files (ical, vtodo, vcard).

To export for example a vcard, connect via HTTP(s) to the following URL:
http://[SLOXIP]/servlet/webdav.[MODUL]/[SLOXID]

[SLOXIP] should be replaced with the name or IP-Address from SLOX
[MODUL] should be contacts, calendar or tasks
[SLOXID] should be the unique id on the SLOX

To import a file, connect with a HTTP(s) to the URL and send the file via PUT:
http://[SLOXIP]/servlet/webdav.[MODUL]/[FILE]

[SLOXIP] should be replaced with the name or IP-Address from SLOX
[MODUL] should be contacts, calendar or tasks
[FILE] should be the file to import

Info: It is not possible to update a record on slox with this special files.

10.Documents

The SLOX WebDAV Interface for documents provides a workflow to manage uploaded documents. Management of folders is for organizing documents.

The main focus for implementing SLOX WebDAV Interface for documents is set on a direct filesystem support, as it can be found in the most operating systems. Folders and documents can be created, renamed, copied and moved like on a local filesystem.

The documents area of SLOX has many more functions than the WebDAV protocol supports. Some functions are not accessible through SLOX WebDAV Interface.

The communication protocol of SLOX WebDAV Interface for documents works according to the explanations of RFC 2518.

The following operations are supported through SLOX WebDAV Interface for documents:

- creating a document or folder and overwriting a document
- renaming a document or folder
- downloading the latest version of a document
- deleting a document or folder

The following clients are tested and known to work:

- Network Environment of Windows 2000 and Windows XP
- Applications from the Gnome project that use the gnome vfs library for file access
- Applications from the KDE project that use the kio library for file access
- OpenOffice 1.1.0
- cadaver

The following parts of a document can not be created, modified or deleted with the SLOX WebDAV Interface:

- the title of a document. If a new document is created, the filename will be used for the title.
- the description of a document
- the mimetype of a document. If the mimetype of a document can be identified, it will be saved for that document.

The access rights of documents and folders are set automatically and can not be set manually. Examples:

- if a new folder or a new document will be created it has the same access rights as the folder it is created in
- in all other cases the access rights are not changed.

Use the web frontend to change access rights instead of.

If you overwrite a file in a normal filesystem, the overwriten file will be lost. Overwriting a document with the SLOX WebDAV Interface saves the overwriten document as an older version and creates a new version of the document. But it is impossible to download any older version of a document. This can be only done with the web frontend.

11.Example Applications

For example source code and some applications please visit
http://devel.slox.info

12.Other References

- WebDAV
  http://www.webdav.org
- RFC 2518
  http://www.ietf.org/rfc/rfc2518.txt
- iCAL
  http://www.ietf.org/rfc/rfc2445.txt
- vCard
  http://www.ietf.org/rfc/rfc2426.txt
- vTODO
  http://www.ietf.org/rfc/rfc2446.txt